Please delete the paragraph at page 4, lines 16-31, and enter the following replacement paragraph:

The monitoring station 100 includes a Service Monitor (SM) 110, which may have a central processing unit (CPU), memory, user interface, such as a command line interface or GUI, and a properties file. The station 100 may be a workstation or other known computer device that operates using an operating system or platform, e.g., such as Solaris™/Linux® or Microsoft Windows® (NT/2000). Software instructions may be stored locally to the station 100 for executing by the CPU in a known manner. The SM 110 may receive profile information, discussed further below, from a profiles store 130, which in turn communicates with a SM server 150. The SM 110 stores test data results in a datalog 140, which also communicates with the server 150. In one possible embodiment, the SM Server 150 may be a 100% pure Java server to support the SMs by performing datalog management and profile management. In addition, the server 150 may distribute profiles to multiple monitoring locations, and the reporting data feeds obtained from the monitored locations to individual customer locations. The profiles 130 include a database of customer monitoring requirements, e.g., the parameters associated with the dial-up point or site that are of interest to the customer. The datalogs 140 include raw performance and service availability measurements recorded by the monitors.

Please delete the paragraph at page 5, lines 12-17, and enter the following replacement paragraph:

In addition to testing the availability, data rate and the like of the dial up points, the SM 110 may run a transaction monitoring process which acts like an end user of each service at the site 170 to measure the service's current status. The transaction monitor process can combine and manage a sequence of other monitors, e.g., to simulate the actions of a real user. Events are fed back into the locally managed datalogs 140, and optionally in real-time to an end user location.

Please delete the paragraphs at page 10, lines 14-28, and enter the following replacement paragraphs:

The SMs may be implemented as a suite of modular data collectors designed to run with Micromuse's® flagship Netcool/OMNIbus® system, available from Micromuse Inc., San Francisco, California, or with other appropriate products. Netcool/OMNIbus® is a client-server application based on the Object Server, which normalizes and synchronizes these events into a common format. This allows operators to custom design service views on-the-fly. Using Boolean filters, views are created based on which events affect the availability of user-oriented services. Probes, which are passive software modules, collect event data of hundreds of applications environments, management systems, and devices.

In Netcool's® hierarchical model, a service is comprised of lower-level, more granular services. These services include key IP protocols, such as those tracked by the Service Monitors. Likewise, a service level is a linear set of hierarchical services. Events pushed into the Object Server by the Probes or Service Monitors define services in terms of binary status, e.g., "Good," "Marginal," "Unknown," "Bad," etc. These are correlated with the underlying network events such as "Link Down," "Process Down," "File System Above Threshold," "Application Failure," and so on.

Please delete the paragraphs as page 12, line 15 to page 13, line 2, and enter the following replacement paragraphs:

In a standalone mode, a number of dial monitor processes are started. The processes may be threads or instances, for example, depending on the platform used. Each dial monitor process parses a profile and opens a connection to a dial-up point. The specific procedure for establishing the connection will vary depending on the platform. A Solaris™/Linux® implementation of the invention may spawn another process, such as a PPP daemon, which actually establishes a connection and a PPP link with the dial up point as a POP of the site. Such a daemon may also handle authentication with the dialed location. Alternatively, a Microsoft Windows® (NT/2000) implementation of the invention may use a Microsoft Remote Access Service (RAS) Application Programming Interface (API) instead of a daemon, as this provides the developer with the interface to gather all the data the service monitor requires. Moreover, the use of this API allows for other changes to the monitor architecture detailed further below.

In the standalone mode, the Windows®-based monitor obtains configuration information based on profile entries. Once profiles are read, the Dial Monitor process spawns a thread for each "dial-up" connection. Each "dial-up" connection is defined by a "modemdevice" field in a profile entry. This also exists in a Solaris™/Linux® implementation, but a modemdevice command-line option or property entry is required to relate one to the other. The binary will be at any time running as many connection threads as there are different comms (communications) ports (as defined by "modemdevice" field entry) defined by profile entries.

Please delete the paragraph at page 13, lines 9-17, and enter the following replacement paragraph:

The behavior of each dial monitor process may be based on profile entries, and also controlled by "properties", which can be entered either at command-line or read in from a properties file. Each monitor is capable of supporting multiple profiles. Note that the term "process" or the like is meant to encompass instances, such as used with Solaris™/Linux® platforms, as well as threads, such as used in Windows platforms. To enable multi-modem support, the dial monitor uses an additional property, "modemdevice", which is also mirrored in an additional field in the dial monitor profiles. This is the case for the Solaris™/Linux® based implementation. For the Windows® based implementation, the "modemdevice" property is not needed or used.

Please delete the paragraph at page 13, lines 27-30, and enter the following replacement paragraph:

On Windows®, the situation differs in that, as there is a single binary, profiles are read/parsed and a connection thread is created for each "modemdevice" profile entry. There is no need to try and synchronize multiple instances of the dial monitor with corresponding profile entries.

Please delete the paragraphs at page 14, lines 13-26, and enter the following replacement paragraphs:

Or, depending on the platform that is used, a properties file could be used instead of the command line entries. For example, a command line interface may be used with a Unix® platform, while a properties file may be used with a Windows® platform.

It should be appreciated that various operating systems may be used, including, e.g., Sun Solaris™, Linux®, Windows NT® and Windows 2000®. The code is used to command a communications port, and as understood by those skilled in the art, can vary based on the operating system, specific equipment used, and so forth.

In the above Solaris™-specific syntax, /dev/cua/a, etc. are just port names, which will be different, e.g., on Linux®. A Windows®-based implementation of the invention, in one possible implementation, does not need this, but may use a syntax as follows:

./nco_m_dial –modemdevice <comms port 1> –datalog &

./nco_m_dial –modemdevice <comms port 2> –datalog &

./nco_m_dial –modemdevice <comms port 3> –datalog &

where <....> indicates changeable syntax.

Please delete the paragraph at page 14, line 29 to page 15, line 4, and enter the following replacement paragraph:

The standalone process is summarized at blocks 300 through 360 in FIG 3. The steps shown need not necessarily be performed in the order given. The multiple dial monitor processes are started (block 300) sequentially, concurrently, or in any other way. For example, for a Windows®-platform based implementation, the processes may be threads that are started substantially in parallel. The dial monitor processes establish respective connections to dial up points of a network site using the designated communication port (block 320).

Please delete the paragraph at page 15, line 29 to page 16, line 4 and enter the following replacement paragraph:

After the transaction monitor process spawns the dial monitor processes as child processes in its dial-up step, it informs them that it is running in the transaction mode. This may be achieved, e.g., by passing command-line arguments to the dial monitor processes. The same applies to a Windows® implementation as to a Solaris™/Linux® implementation, but with Windows®, "modemdevice" property is not passed. Thus, the transaction monitor process is now modified to pass an additional command-line argument – "modemdevice"- with a value that corresponds to the "modemdevice" profile entry in the dial-up step.

Please delete the paragraph at page 16, lines 21-26, and enter the following replacement paragraph:

The transaction monitor has analogous functionality (as it appears to the user) as the Solaris™/Linux® version. One difference is that, when a dial process is spawned by the transaction monitor process, it will then spawn a connection thread (in Dial UP), or will signal the running dial monitor to shut down (on Dial Down). In a Solaris™/Linux® implementation, the dial monitor process spawns a PPP daemon process (on Dial UP) or signals the running PPP daemon process to shut down (on Dial Down).